Detecting and reacting against distributed denial of service attacks

Yacine Bouzida, Frédéric Cuppens, Sylvain Gombault

Detecting and Reacting against Distributed Denial of Service Attacks

Yacine Bouzida
Mitsubishi Electric ITE-TCL
1, allée de Beaulieu CS 10806
35708, Rennes, France
Bouzida@tcl.ite.mee.com

Frédéric Cuppens and Sylvain Gombault
Département RSM GET/ENST Bretagne
2, rue de la Châtaigneraie F-35576, Cesson Sévigné, France
Frederic.Cuppens@enst-bretagne.fr

***Abstract*— Distributed denial of service attacks (DDoS) are becoming a big threat to the Internet. Recently, some DDoS attacks have infected more than $100,000$ vulnerable hosts over Internet within 10 minutes. Consequences of these attacks can be devastating toward many companies whose security policy against this kind of attacks relies only on reconfiguring firewalls. It is judicious to note that no computer network is immune from intrusions in general and distributed denial of service attacks in particular. Intrusion detection systems should be geographically distributed to detect distributed and cooperated attacks. In this paper, we use a cooperative approach, which uses the Intrusion Detection Message Exchange Format (IDMEF) defined by the IETF, that can detect coordinated attack scenarios through alert correlation of distributed IDSs. We present our experience in realizing this cooperative system and the different results obtained from its implementation in a real network.**

I. Introduction

Current Computer networks are designed with functionality where security is not considered as a main goal. For this reason, Internet is offering clients fast, easy and cheap communication mechanisms, at the network level, that provide best effort service to the different designed protocols. These services are implemented at the end points; the sender and the receiver. With this implementation, malicious end users can easily violate the described policy of the different protocols and act to damage the other party. In this case, the end-receiver resources might be overwhelmed by the bad traffic sent by the malicious end-sender-point.

Recent DDoS attack tools such as Trinoo, Stacheldraht, etc. use forged addresses to flood a victim, which might be any entity connected to Internet and providing some service to other legitimate users, in order to decrease or disable the service provided by this entity. Ingress and Egress filtering are introduced to prohibit attackers from originating network to launch attacks with forged addresses. These filtering techniques cannot stop forged packets with the same prefix as that of the network from where they are originated. However, it enables the originator to be easily traced to its true source, since the attacker would have to use a valid, and legitimately reachable, source address. Filtering preventive techniques are necessary to reduce DoS attacks using spoofed addresses but are not sufficient enough to stop definitively DoS or DDoS attacks.

For this reason, we propose to use intrusion detection as another barrier to counter these distributed attacks. The idea consists in stopping the attack during its first stage.

A cooperative intrusion detection framework collecting alerts from many trusted points over Internet and communicating attack information between the different IDSs is of a great interest to detect any distributed denial of service at its early stage. Once an attack, which corresponds to the first step of a global attack, is detected by an IDS, other IDSs, geographically located elsewhere over Internet, may take some advantage and launch a counter measure against the compromised host to stop the ongoing attack. In fact, DDoS attacks correspond exactly to this situation. As described in Dittrich [8], the DDoS attacks require to compromise and recruit hundreds (or thousands) of slaves over the web and other hosts called masters which will play the relay between the attackers and the slaves in order to launch their attacks against a victim using this architecture.

The remainder of the paper is organized as follows. Section II describes the standard DDoS tools. Section III presents alert correlation in intrusion detection followed by section IV in which we model the standard DDoS tools in LAMBDA [6]. Finally, our experimentation is described in section V and section VI concludes the paper.

II. Description of the flooding DDOS tools

The main architecture of classical DDoS attacks, based on flooding, is presented in Figure 1.

The attacker starts a session with the master host. The daemon process, launched in the master host, offers many commands facilities to the attacker in order to launch the desired flooding attacks.

After establishing a connection with the master, the attacker may launch a flooding attack against one (or many) victim(s) using the different commands offered by the master. The master receives the commands through the established connection with the attacker and sends the corresponding commands to the slave(s) using the specified communication with the slave. Hence, the slave floods the victim using the specifications received from the master. We mention that the daemon installed on the slave machine sends a message to the master(s) to inform it (them) that it is alive. Note that due

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

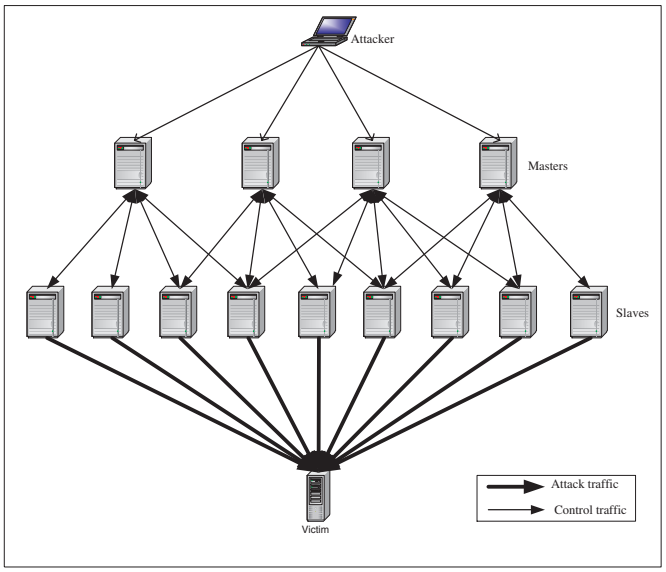

Fig. 1. DDoS Attack Architecture.

to space limitaion, we do not give a thorough description of DDoS attacks, many references may be found at [8].

III. Alert Correlation in intrusion detection

Correlating information held by multiple intrusion detection alerts is an approach that has been discussed in several papers ([13], [7], [9], [5], [3], [10], [11]). However the goal aimed by those approaches are different and needs to be explicited.

With the rise of cooperative or distributed intrusion detection framework, the problem of reasoning on information coming from multiple sources spread across the monitored system is very important. Correlating that information allows to fulfil different goals. Here are some problems addressed by alert correlation:

- *Information redundancy*: a set of IDSs distributed across a network increases the intrusion detection power. But when an event is detected, it can be detected by a subset of this set of IDSs. Hence, we may obtain a set of alerts for the occurence of an event. Those alerts related to the same event must be aggregated and fused to obtain a more synthetic alert for further processing.
- *Scenario detection*: the granularity of intrusion detection alerts is low. The event associated to an alert can be non malicious when considered alone but when a more global vision is adopted we may conclude that this elementary event is part of an ongoing intrusion scenario.

The notion of alert correlation as a process of aggregating alerts related to the same event has been studied in [13], [7], [2]. Those papers define a similarity relationship between alert attributes and use this relationship to aggregate alerts. The second main approach of alert correlation as a process of detecting scenarios of alerts has been discussed in [10], [3], [1].

In this paper we use the notion of alert correlation as the process of finding a set of alerts into the stream of intrusion detection alerts organized into a scenario. Our formalism is exposed briefly in the following subsection. We recall the definition of correlation and anti-correlation as our main goal is to react to the DDoS scenario when we observe the beginning of the scenario.

A. Intrusion modeling

From the intruder point of view, the intrusion process can be seen as a planning activity ([3]). The intruder can have knowledge of the system he/she wants to attack, knowing the set of vulnerabilities or the used set of software and hardware. If the attacker has a limited knowledge about the aimed system, he can try to gather information by executing some actions such as ports scans or using vulnerability detection tools. Once the attacker has a sufficient knowledge of the system, he/she can define a set of reachable intrusion objectives. From the point of view of the victim, those intrusion objectives constitute a violation of the security policy. In order to reach those intrusion objectives, the attacker selects a set of actions constituting one or multiple scenarios of actions.

From the detection point of view, we want to detect the intrusion by constructing scenarios of alerts corresponding to the scenarios of actions executed by the attacker. In order to do so, we have to model the set of actions available for the attacker and a set of intrusion objectives. Since we want to react to the detection of ongoing scenarios, we have to model the set of available counter-measures. We use the LAMBDA language to model the actions, the intrusion objectives and the counter-measures.

LAMBDA is an acronym for LAnguage to Model a dataBase for Detection of Attacks. It provides a logical and generic description of actions, but we use it to model as well the intrusion objectives and the counter measures. Note that the LAMBDA language is not specific to the field of intrusion detection.

A LAMBDA description of an action is composed of the following attributes:

- *pre-condition*: defines the state of the system needed in order to achieve the action.
- *post-condition*: defines the state of the system after the execution of the action.
- *scenario*: the combination of the different events involved in the scenario describing the attack.
- *detection*: describes the expected alert upon the detection of the action.
- *verification*: specifies the condition to verify the success of the action.

The intrusion objectives are defined as a condition over the system state. This state corresponds to a violation of the security policy. Modeling an intrusion objective in LAMBDA requires only one attribute: the system state condition. The following section presents the action models expressed in the LAMBDA language.

IV. Modeling Standard DDoS Tools with LAMBDA

In this section, we present our modeling of the actions involved in the DDoS attack executed thanks to the different standard DDoS tools. We have determined the signatures necessary to detect the communications between the attacker and the master(s) and between the master(s) and the slave(s). Those signatures allow us to detect the full set of actions involved in the realization of a DDoS attack using the Trinoo tool.

We model the following four actions:

- *Opening a connection between the attacker with the master*: the connection used by the attacker to communicate with the master computer represents the beginning of the scenario.
- *"show alive message" message from slave to the master*: the message sent by the slave to notice the master computer that it is ready to receive commands. This action may be performed in parallel with the first step.
- *"dos" command sent by the attacker to the master*: the command sent by the attacker to the master computer to start a DDoS attack using the detected slave(s) computer(s).
- *"dos" command sent by the master to the slave*: the command sent by the master to the slave(s) computer(s) to start flooding the victim(s).

The action models are presented in Figure 2. Note that the DDoS command sent by the attacker modeled as action *command_dos* represents one possible command, the standard DDoS tools allow other actions.

The intrusion objective is modeled in Figure 3. In our example, the DDoS attack makes the victim computer not anymore available.

attack $connection(A, H)$
pre: $remote\_access(A, H)$
$\wedge$ $master(H)$
detection: $classification(Alert,$'$master$'$)$
$\wedge$ $source(Alert, A)$ $\wedge$ $target(Alert, H)$
post: $connected(A, H)$
verification: $true$

attack $show\_alive(S, H)$
pre: $remote\_access(H, S)$ $\wedge master(H)$
$\wedge$ $slave(S)$
detection: $classification(Alert,$'$show\_alive(S, H)$'$)$
$\wedge$ $source(Alert, S)$ $\wedge$ $target(Alert, H)$
post: $knows(H, slave(S))$
verification: $true$

attack $command\_dos(A, H, V)$
pre: $connected(A, H)$
$\wedge$ $master(H)$,
$\wedge$ $knows(H, slave(S))$
detection: $classification(Alert,$'$command\_dos$'$)$
$\wedge$ $source(Alert, A)$
$\wedge$ $additional\_data(Alert,$'$ddos\_victim$'$, V)$
$\wedge$ $target(Alert, H)$
post: $dos\_command\_sent(H, S, V)$
verification: $true$

attack $command\_dos\_to\_slave(H, S, V)$
pre: $knows(H, slave(S))$
$\wedge$ $master(H)$
$\wedge$ $dos\_command\_sent(H, S, V)$
detection: $classification(Alert,$'$command\_dos\_to\_slave$'$)$
$\wedge$ $source(Alert, H)$
$\wedge additional\_data(Alert,$'$ddos\_victim$'$, V)$
post: $distributed\_denial\_of\_service(V)$
verification: $unreachable(V)$

Fig. 2. Modeling a DDoS scenario.

objective $ddos(V)$
state: $distributed\_attack(V)$

Fig. 3. Modeling the DDoS scenario objective.

A. Responding to the scenario

Detecting the DDoS scenario is interesting but it does not prevent the attacker from reaching her/his objective. In order to stop the attack we have to find one appropriate counter-measure and we have to launch it before the *DOS* command is issued by the master(s).

We use the $kill$ command that can be issued from the master(s) to destroy one or several slaves. This command must be issued once the master knows all the slaves' computers. The counter-measure model is represented in Figure 4.

Issuing this counter measure on the list of detected slave computers prevents those computers from flooding the victim, but we cannot guaranty that we have detected all the slaves used to achieve the attack, namely when the compromised hosts are not monitored. We could also kill the connection between the master and the attacker by sending a TCP reset but the attacker could easily open a new connection and try again to attack. Killing the slave computers is more desirable since it prevents from further attacks from those computers, until the slave daemon is reinstalled. Other counter measures may be taken by the administrators of the networks where the compromised hosts are located. This will be discussed in section V.

B. Scenario detection

In this section we expose the way the DDoS scenario is detected through the use of the notion of correlation as defined in [5]. We also explain how we react to the detection of this scenario through the use of anti-correlation as explained in [1].

1) Action correlation: Two actions A and B are correlated when the realization of A has a positive influence over the realization of B (given that A occurred before B). More formally, if $post(A)$ is the set of post-conditions of action A and $pre(B)$ is the set of pre-conditions of action B, we say that A and B are *directly correlated* if the following conditions are satisfied:

$\exists E_a$ and E_b such that:

- $(E_a \in post(A) \wedge E_b \in pre(B))$ or $(not(E_a) \in post(A) \wedge not(E_b) \in pre(B))$
- E_a and E_b are unifiable through a most general unifier

counter_measure $kill\_slave(S)$
pre: $slave(S)$
action: $kill\_slave(S)$
post: $not(slave(S))$
verification: $not(slave(S))$

Fig. 4. Modeling the DDoS scenario counter measure.

θ.

Similarly, we define the notion of correlation between an action and an intrusion objective. In this case we correlate the post-condition of an action and the state condition of an objective.

2) Alert correlation: Once all the actions available for the attacker have been modeled, we can generate the set of unifiers between all the actions. This generation is done off-line. When an alert is received, we have to bind this alert to an action model and then check a unifier between the new alert and the already received alerts. This set of unifiers is also used to anticipate the possible actions we may see after having observed the beginning of a scenario. Those hypothetic observations are called $virtual\ actions$.

3) Action anti-correlation: Two actions A and B are anti-correlated when the realization of A has a negative influence over the realization of B (given that A occured before B). More formally, if $post(A)$ is the set of post-conditions of action A and $pre(B)$ is the set of pre-conditions of action B, we say that A and B are *directly anti-correlated* if the following conditions are satisfied:

$\exists E_a$ and E_b such that:

- $(not(E_a) \in post(A) \wedge E_b \in pre(B))$ or $(E_a \in post(A) \wedge not(E_b) \in pre(B))$
- E_a and E_b are unifiable through a most general unifier θ.

In our scenario example, once the master starts to receive the $showalive$ message from the slaves, we can identify the slave computers and kill them by launching our counter-measure.

Figure 5 represents the full correlation graph for the DDoS scenario. Note that once we know all the slave computers, we can launch the counter measure and it will be effective until the $command\_dos\_to\_slave$ action is launched.

V. Experimentation

We used our campus network to test our correlation engine to detect the different scenarios and to react against them. We used four different subnets where signature based IDSs are present in each subnet. We did not use anomaly detection tools because the different elementary attacks composing the DDoS attack cannot be detected with such tools. In fact, the communications between the different components (attacker(s), master(s) and slave(s)) correspond, in reality, to a normal traffic as described above, so an anomaly detection tool cannot detect the communications between the different components as preliminary steps of the DDoS attack. However, an anomaly detection tool, in front of a victim, may deal with this kind of attacks. In this case, it is too late to react against it. Our principal goal is to stop an ongoing denial of service attack before any victim undergoes considerable damages. Recently, a company (see for instance http://seclists.org/lists/isn/2004/Jul/0106.html) was downed by a denial of service attack, during four complete hours, targeting its DNS servers. This attack was detected at the victim but nothing could be done at this stage where it has lasted many hours. It is preferable that this attack were countered before it caused its negative effects towards the DNS servers. The outside sources of the attack were unknown because of IP Spoofing.

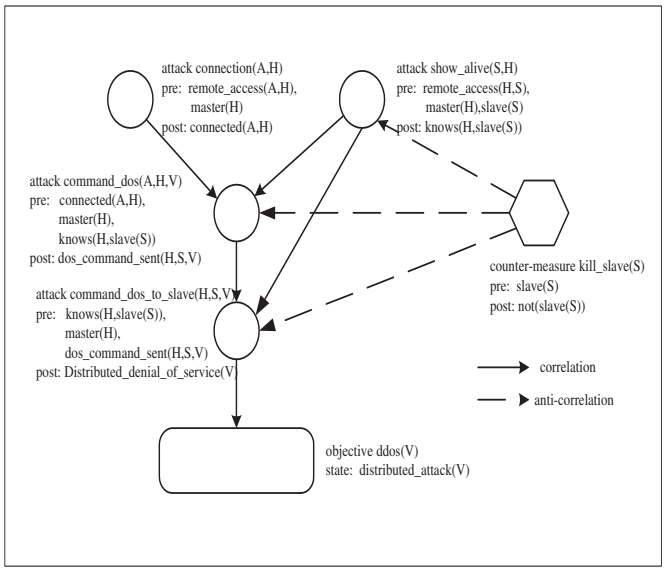

Fig. 5. DDoS Attack correlation graph.

In our experiments, the attacker(s)[1], the master(s), the slaves and the victim(s) are located in different subnets as if they were in different geographically distant networks over Internet.

DIAMS[2], a platform we have developed in Java to corporate the different intrusion detection tools, is installed in another subnet. DIAMS framework is described in Figure 6.

In the following, we present the different modules composing this platform and their corresponding functions. First, DIAMS collects syslog alerts sent by the different sensors that are installed in the different subnets. A translator module receives the syslog alerts and translates them into IDMEF (Intrusion Detection Message Exchange Format) format. These alerts are then sent by a module called *router* to an alert database (managed by PostgreSQL) and to the CRIM engine where new functionalities [1], such as anti-correlation [1] and weighted correlation [1], have been added to the first version [4], [5], [2]. The new aggregation/fusion module, in opposite to the first version [2], has been modified to use a similarity function that calculates the similarity value (a real number in $[0, 1]$) between two alerts. The first version was using a prolog implementation which was not able to assess the level

[1]no IDS tool is installed in the attacker subnetwork.

[2]DIAMS stands for Détection d'Intrusion Au Moyen de Sondes

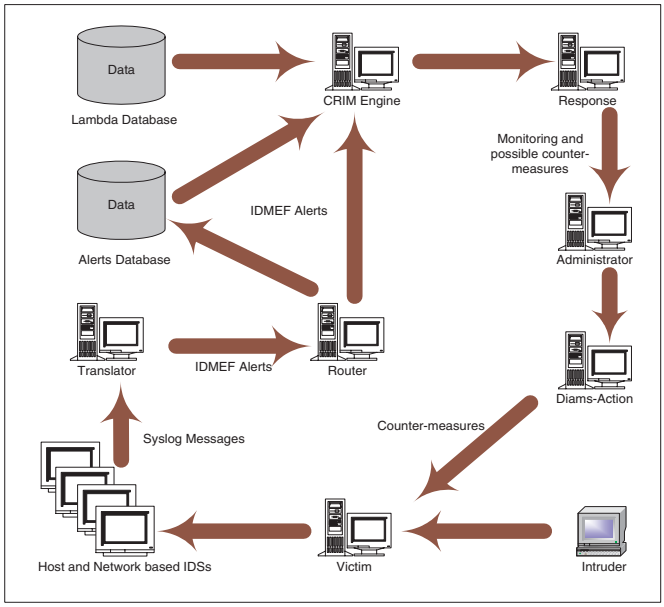

Fig. 6. DIAMS Framework.

of similarity between two alerts, two alerts were considered as similar or not.

After the aggregation and the fusion procedures, CRIM correlates the different aggregated alerts using the method presented in section III and generates a diagnosis of the detected intrusion scenarios. This diagnosis is transmitted to the response mechanism that provides the concerned administrator with a set of candidate counter-measures using anti correlation defined in section III. The administrator where th victim is located can then select one or several counter measures. DIAMS offers, through a module called *DIAMS-Action*, the possibility to launch a response to stop an ongoing attack by sending a TCP Reset or by sending other packets for example to kill a slave or a master in our case. However, this module is installed in each subnet but not in the attacker subnet because actually we cannot react against attackers, since we do not know where they are and we do not have sufficient information about their network or computer system. In addition, some counter measures are not possible because of some filtering restrictions. For all of these reasons, DIAMS-Action is installed in the *victim* side that we want to protect.

We illustrate the DDoS attack scenario described in Figure 5. As mentioned in section II, we do not take into account the early steps of the DDoS scenario that consist in scanning and compromising the different hosts that will later play the role of masters and slaves. These intrusions may be detected with some IDSs and then some corresponding scenarios may be constructed with a semi explicit manner which may be detected by the correlation engine. However, if the attacker has a physical or remote access to the different machines then these first steps cannot be detected.

The first step consists in opening a connection between the attacker with the master. The IDS installed in the masters' subnet will generate an alert and sends it to the DIAMS platform. The pre-condition of the first attack of the scenario mentions that the H host is a master host compromised by the attacker A. Its post-condition specifies that the master has opened a connection with the attacker.

When the attacker launches (automatically or not) the daemon(s) in the different compromised slaves hosts, these slaves hosts will send a *"show alive"* message to the masters that control them in order to inform their readiness to flood. Hence, the IDS situated in the subnetwork containing the master(s) host(s) will send an alert (or alerts) to DIAMS which in turn translates them into IDMEF format and sends them to the CRIM engine that constructs the first two steps of the scenario. We should mention here that this second step of the global scenario is detected in both the master and slave subnetworks. In this case, the IDS installed in the slave subnetwork will also send the same alert to DIAMS. These two alerts will be easily merged into one single alert by the CRIM engine (for more details on merging, see for instance [2]).

With the same manner, the last two attacks of the scenario will be detected successively by the master subnetwork IDS and the slave subnetwork IDS. The last scenario attack may also be detected by the master subnetwork. It will be aggregated to one alert as the second step.

Once the CRIM engine has recognized the global scenario using the approach explained in the previous sections, the response module launches automatically a counter measure which consists in killing the different slaves processes running in the different hosts located in the slaves subnetwork. In fact, DIAMS sends a message to the DIAMS-Action, which is situated in the subnetwork where the slaves are detected (the slaves subnetwork in our case), to kill the slave daemons. After receiving this message, DIAMS-Action will launch the appropriate response which consists in killing the slave daemon processes in the hosts specified by the DIAMS platform. In addition to this, the administrator where the daemon is located is warned, by the DIAMS platform, about the host where the daemon is installed. He should find the vulnerability that permitted the installation of the daemon on that host and disinfect and patch the system. Without doing this, other daemons may be launched automatically from the same host. This is what is called counter counter-measure.

The attacker believes the success of her/his attack. However, our framework has already killed all the slaves she/he has recruited. We mention that other counter-measures that consist in aborting a TCP connection between the attacker and the master and killing the client program on the master host are also implemented. This last response should be launched but the warned administrator by other informative responses sent by the DIAMS platform should take new security considerations on her/his vulnerable network.

The communications between the different IDSs, the DIAMS platform and DIAMS-Action should be encrypted to not allow any alteration of the exchanged messages by any attacker that can do so. A Public Key Infrastructure (PKI) is implemented in our architecture to solve this problem in

particular when this infrastructure is deployed over Internet.

VI. Conclusion and future work

We have presented in this paper a cooperation technique based on many IDSs tools geographically distant to coordinate their alerts in order to detect a coordinated attack using the CRIM engine. CRIM offers many features such as aggregation, fusion, correlation and anti-correlation to recognize new attack scenarios and launch appropriate counter-measures to stop any ongoing attack that may affect a targeted victim.

The different DDoS tools such as Stacheldraht, TFN, TFN2K, etc. we have investigated are detected by our correlation module with the same manner as the one discussed to detect Trinoo. However, since some communications between the different components in these tools are encrypted, the detection signatures of the encrypted messages are more complex to specify. For this reason, we use a new idea that assigns a confidence ratio to alerts and then a confidence ratio is derived for the ongoing scenario from the confidence ratios of the different alerts composing the scenario. This will be discussed in a forthcoming paper.

Some of counter measures, such as killing the daemons of slaves and masters, are launched automatically to prevent the occurrence of a DDoS attack. However, in all cases, the administrator is provided with a support to take a final decision in order to reconfigure the security policy to prevent a new occurrence of a given intrusion in particular compromising the different hosts which played the role of slaves and masters. However, as suggested in [12], dynamic changes of the security policy may cause failure of some software components. This is why [12] suggests the notion of security agility, a strategy to provide software components with adaptability to security policy changes. Security agility might be nicely included into the intrusion detection and response framework suggested in this paper. This represents a possible extension of the response mechanism in our framework.

Acknowledgment

This work was funded by the GET Crédit Incitatif DDoS project. The authors thank all the members of this project.

References

[1] F. Autrel, F. Cuppens, and S. Benferhat. Enhanced Correlation in an Intrusion Detection Process. In *MMACNS*, St Petersburg, Russia, September 2003.

[2] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *17th Annual Computer Security Applications Conference New-Orleans*, New-Orleans, USA, December 2001.

[3] F. Cuppens, F. Autrel, A. Miège, and S. Benferhat. Recognizing malicious intention in an intrusion detection process. In *Second International Conference on Hybrid Intelligent Systems (HIS'2002)*, pages 806–817, Santiago, Chile, October 2002.

[4] F. Cuppens, S. Gombault, and T. Sans. Selecting Appropriate Couter-Measures in and Intrusion Detection Framework. In *17th IEEE Computer Security Foundations Workshop*, pages 78–87, Pacific Grove, CA, June 2004. IEEE Computer Security.

[5] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE Symposium on Security and Privacy*, Oakland, USA, 2002.

[6] F. Cuppens and R. Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In *Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, Toulouse, France, October 2000.

[7] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Fourth International Workshop on the Recent Advances in Intrusion Detection (RAID'2001)*, Davis, USA, October 2001.

[8] D. Dittrich. Distributed Denial of Service (DDoS) Attacks and Tools. Available at: http://staff.washington.edu/dittrich/misc/ddos/.

[9] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A Formal Data Model for IDS Alert Correlation. In *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002*, pages 115–127, Zurich, Switzerland, October 2002.

[10] P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. In *proceedings of the 9th ACM conference on Computer and communication security*, pages 245–254, Washington DC, USA, 2002.

[11] P. Ning and D. Xu. Learning Attack Strategies from Intrusion Alerts. In *proceedings of the 10th ACM conference on Computer and communication security*, pages 200–209, Washington DC, USA, 2003.

[12] M. Petkac and L. Badger. Security agility in response to intrusion detection. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC)*, New-Orleans, LA, 2000.

[13] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Fourth International Workshop on the Recent Advances in Intrusion Detection (RAID'2001)*, pages 54–68, Davis, USA, October 2001.